Improving Your Application with the Xcode Static Analyzer

Session 401

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Improving Your Application with the Xcode Static Analyzer

Ted Kremenek
Architect, Xcode Static Analyzer

What You'll Learn

- New feature in Xcode to automatically find bugs
- Works on iPhone and Mac applications

What Is Static Analysis?

- Automatically find bugs by analyzing source code
- Static = “don’t run the program”
- Employs deep compiler analysis
- Advanced compiler warnings

Key Strengths of Static Analysis

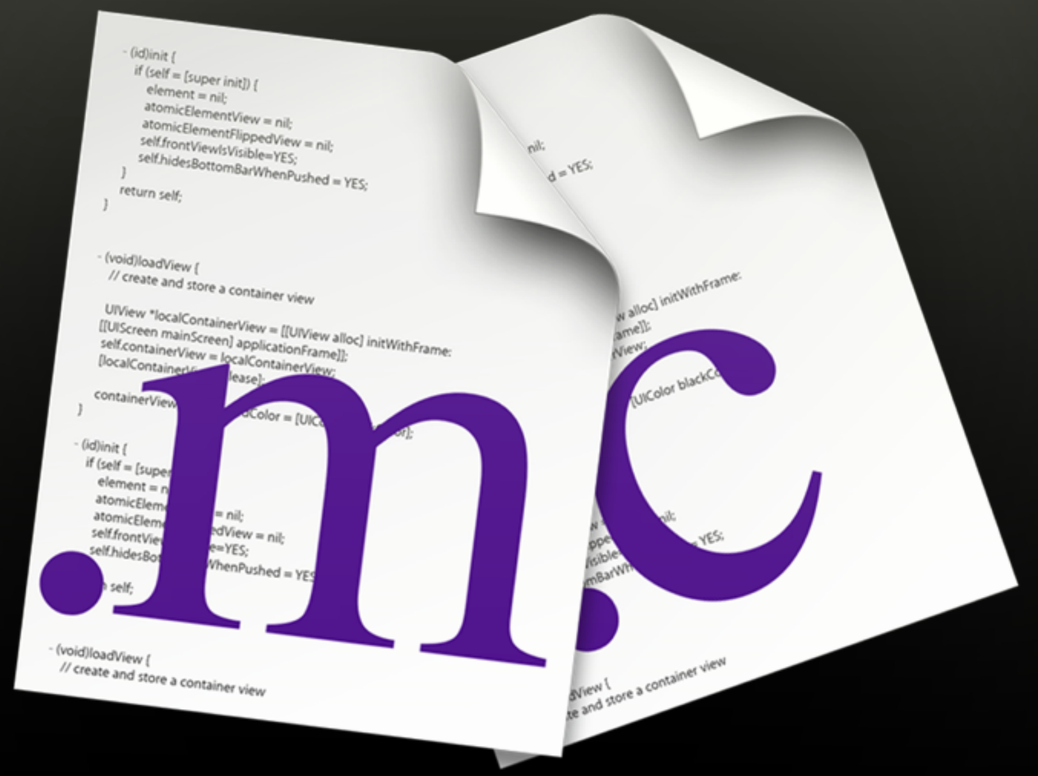
- Early detection of bugs
- Find bugs without test cases
- Detailed error reports

Static Analysis in Xcode

- New in Snow Leopard
- Analyze both iPhone and Mac apps
- Focus areas
  - Cocoa API enforcement
  - Memory management
  - Logic errors
- Uses Clang static analysis engine
  - 100% Open Source
- Fluid workflow within Xcode

Why Should You Care?

Thousands of bugs fixed



Learn and reinforce good Cocoa practices

Fix bugs before users see them

Demo

Yuji Akimoto
Senior Xcode UI Engineer

Roadmap

- How it works
- Xcode workflow
  - Build integration
  - Navigating results
- Bug reports
  - Kinds of checks
  - Dealing with false positives

How It Works

Built on Top of Clang

- 100% Open Source
- Analyzes C and Objective-C code
- Previewed at last year's WWDC

http://clang.llvm.org

Static Analyzer Construction

Clang Compiler

Xcode Static Analyzer

How Is My Code Analyzed?

- Each function/method analyzed one at a time
- Analyzer reasons about possible execution “paths”
- Analyzer abstracts away program details to achieve coverage
- Static analysis is slower than compilation
  - Worst-case exponential algorithms
  - Best effort

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

`result` is declared without an initial value

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

The value of `addr->sa_family` is unconstrained

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

Assuming the value of `addr->sa_family` is `AF_INET6`

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

Call to ‘inet_ntop’ returns a ‘const char*’

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

Assuming return value is null. Taking false branch

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, ..., buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;
}
```

`result` is still uninitialized. Garbage returned to caller

Anatomy of a Bug

```
static NSString *getSocketAddress(const struct sockaddr *addr)
{
  NSString *result;                                  Variable 'result' declared without an initial value
  char buffer[32];
  switch (addr->sa_family) {
    case AF_INET:
      if (inet_ntop(AF_INET, &((struct sockaddr_in *)addr)->sin_addr, buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    case AF_INET6:
      if (inet_ntop(AF_INET6, &((struct sockaddr_in6 *)addr)->sin6_addr, buffer, sizeof(buffer)))
        result = [NSString stringWithUTF8String:buffer];
      break;
    default:
      result = nil;
  }
  return result;                                     Uninitialized or undefined value returned to caller
}
```

Limitations of the Static Analyzer

Inherent Limitations

- False positives
- Static analyzer will not find all of your bugs
  - Goal is to find bugs, not prove correctness
  - Prefer to miss bugs than produce noisy results

Current Limitations

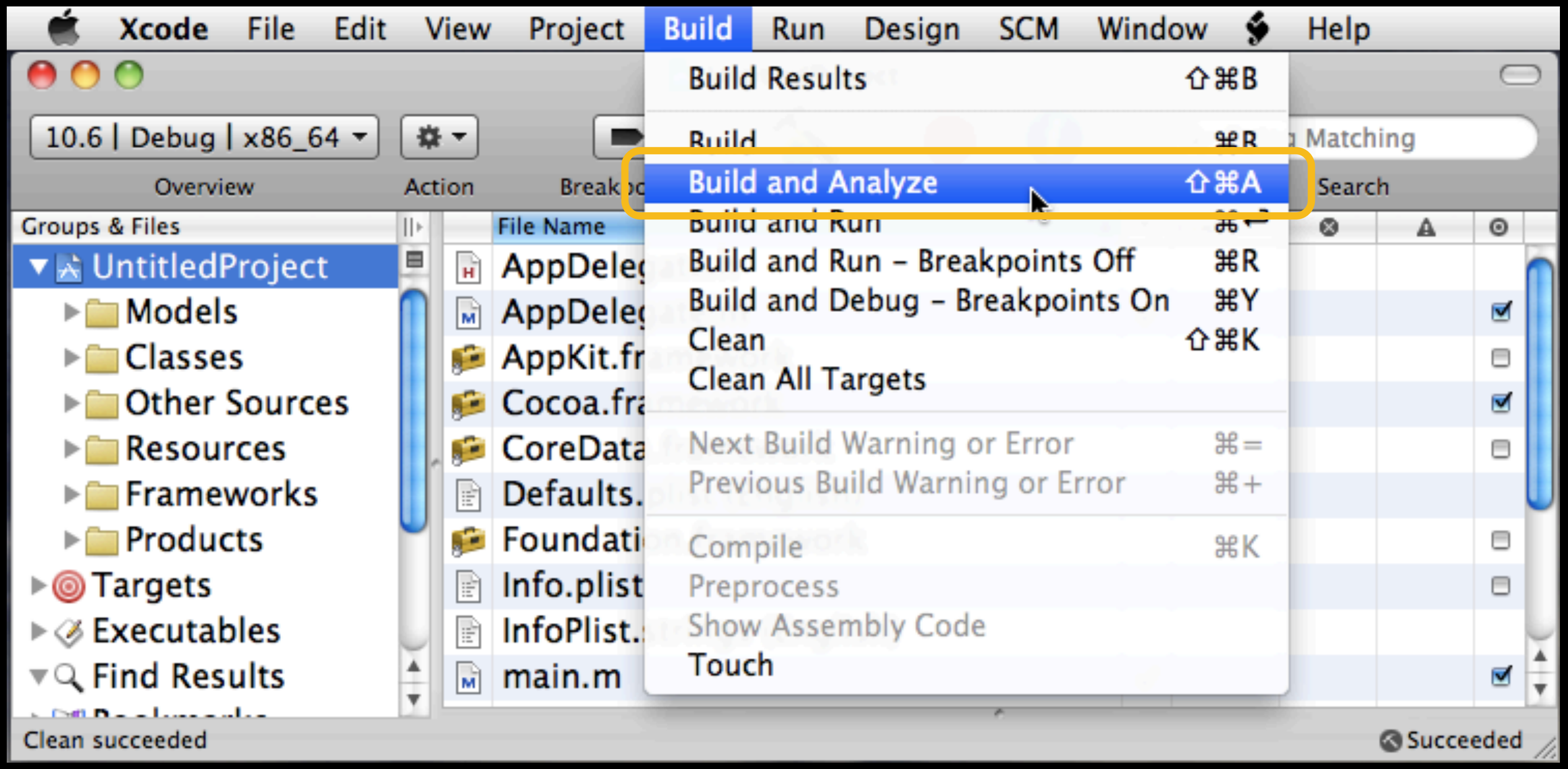
- No analysis across function call boundaries
- No buffer overflow checking

Xcode Workflow

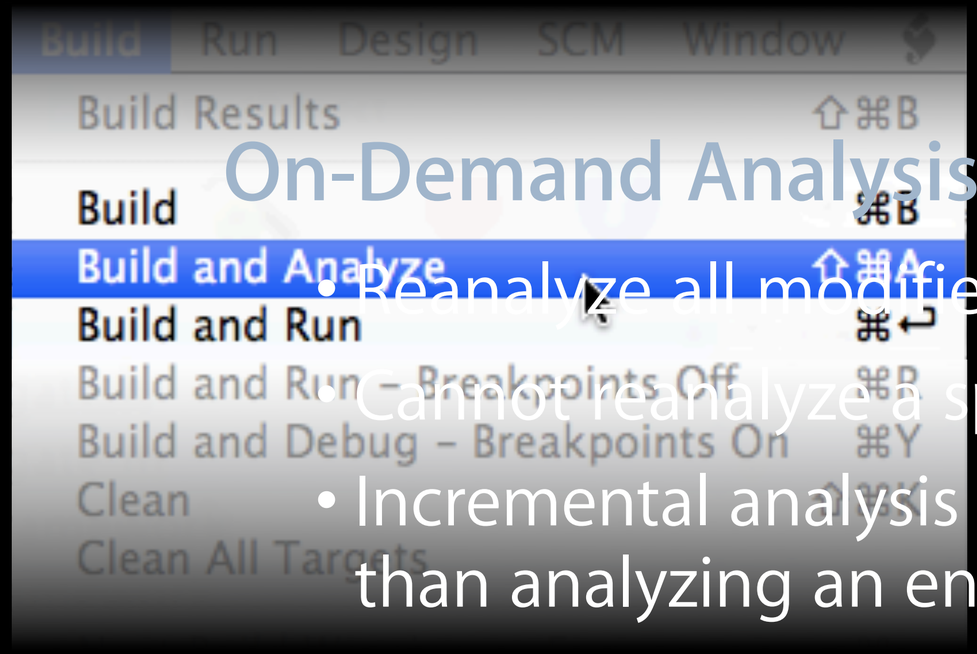
Goals of Xcode Workflow

- Analyze code while actively developing
  - Never leave the editor
- Inspect analyzer issues with regular errors and warnings
- On-demand versus automatic analysis
- Analyze-Fix-Analyze workflow

On-Demand—Build and Analyze



On-Demand—Build and Analyze



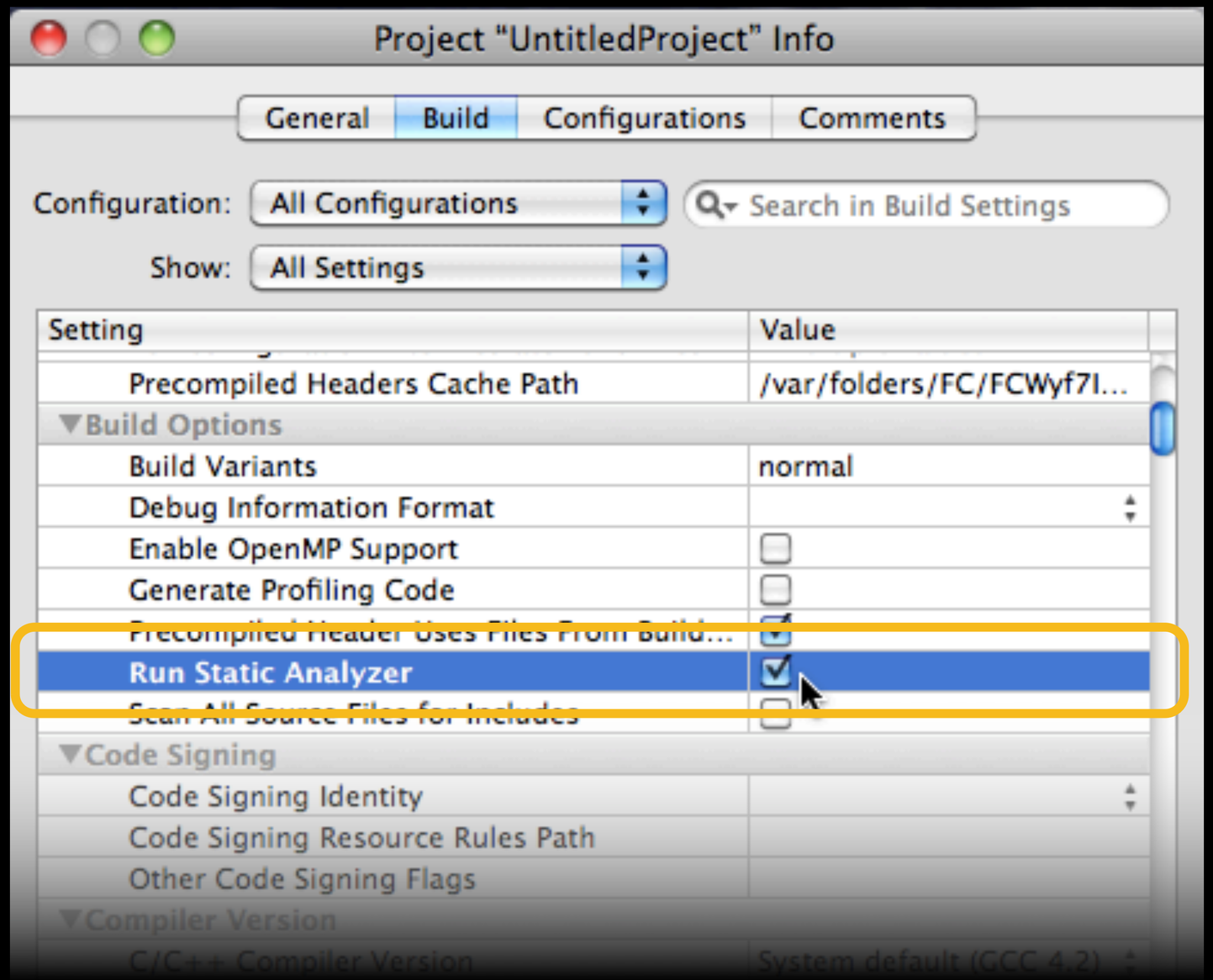
On-Demand Analysis

- Reanalyze all modified files
- Cannot reanalyze a specific file
- Incremental analysis much faster than analyzing an entire project

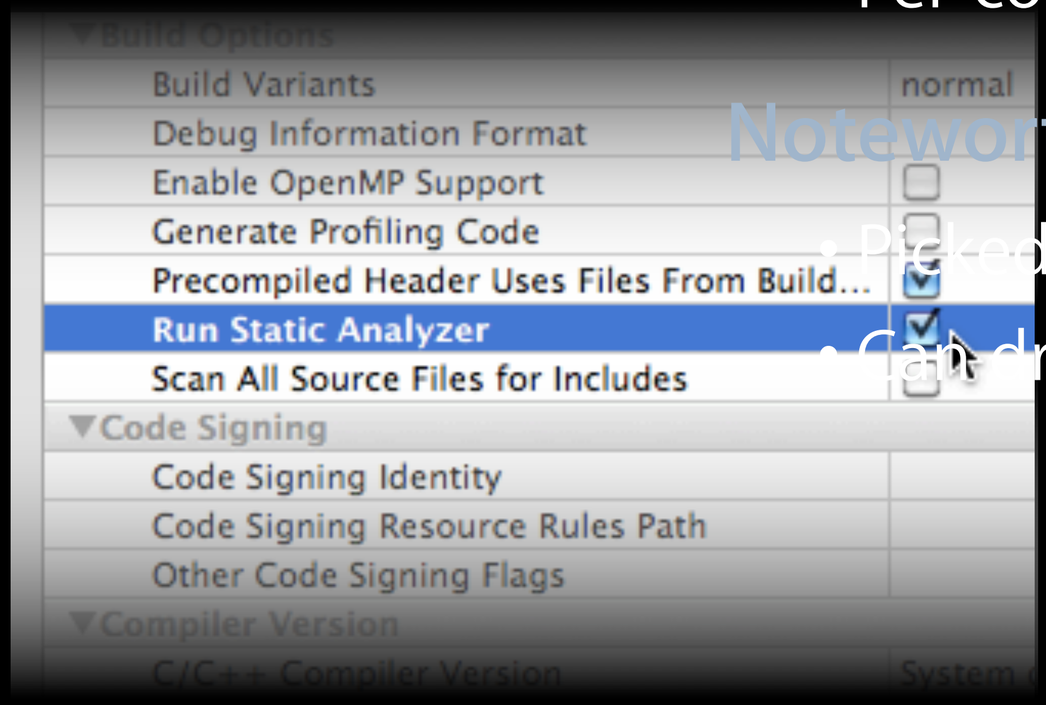
Analyze-Fix-Analyze

- Fix a bug and reanalyze to see if the reported issue disappears

Automatic Analysis as a Build Option



Automatic Analysis as a Build Option



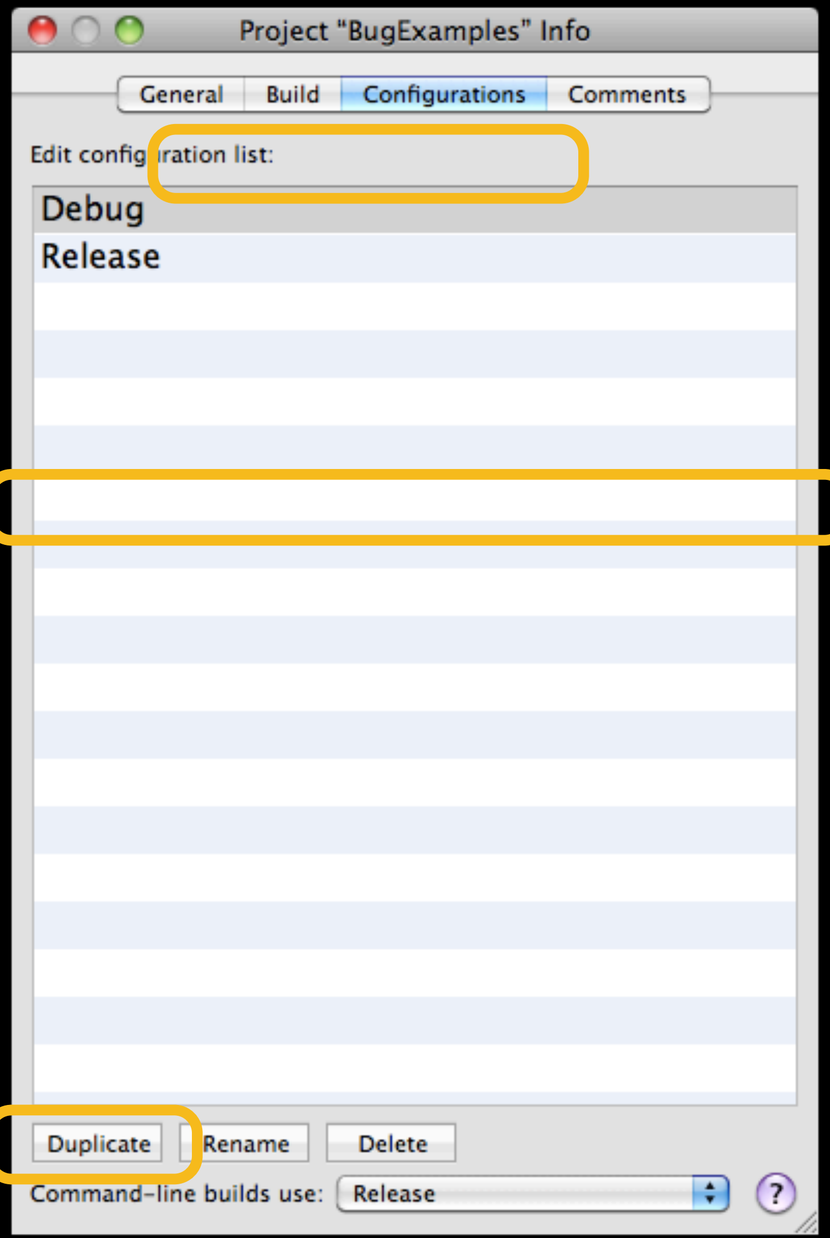
"Run Static Analyzer"

- Always analyze compiled files
- Persistent setting in Xcode project
- Per-configuration setting

Noteworthy Side Effects

- Picked up by xcodebuild
- Can drastically increase build times

Creating an “Analyze” Configuration



Navigating Issues

Yuji Akimoto
Senior Xcode UI Engineer

Bug Reports

Getting the Most out of Analysis Results

- Kinds of checks
  - Cocoa and Core Foundation
    - API checks
    - Memory management
    - General language-level or logic errors
    - Dead stores
- Dealing with false positives
  - Assertions
  - Annotations

CFNumberCreate (64- vs. 32-Bit Issues)

- `CFNumberCreate` is a Core Foundation factory function
- Be careful of 64- vs. 32-bit architecture differences!

```
unsigned i = 10;

CFNumberRef x = CFNumberCreate(0, kCFNumberLongType, &i);

return x;
```

CFNumberCreate (64- vs. 32-Bit Issues)

- `CFNumberCreate` is a Core Foundation factory function
- Be careful of 64- vs. 32-bit architecture differences!

```
unsigned i = 10;

CFNumberRef x = CFNumberCreate(0, kCFNumberLongType, &i);

return x;
```

CFNumberCreate returns a new `CFNumber` object that is initialized with the value of `i` (that is passed-by-reference via a `void*`)

A **32 bit integer** is used to initialize a `CFNumber` object that represents a **64 bit integer**. 32 bits of the `CFNumber` value will be **garbage**.

same size (32-bits) for -arch **i386**

```
unsigned i = 10;

CFNumberRef x = CFNumberCreate(0, kCFNumberLongType, &i);
    A 32 bit integer is used to initialize a CFNumber object that represents a 64 bit integer. 32 b...
return x;
```

long and **unsigned int** have different sizes (64- and 32-bits) for -arch **x86_64**

Cocoa Memory Management

Only on Mac OS 10.5+

- Analyzer checks Cocoa object ownership conventions
  - *Memory management guide for Core Foundation*
  - *Memory management guide for Cocoa*
- Garbage collection versus retain counts
  - *Garbage collection programming guide*

Flag	Mode
-fobjc-gc	Use either garbage collection or retain counts
-fobjc-gc-only	Use only garbage collection
No flag	Use only retain counts (traditional mode)

Cocoa Naming Conventions

Cocoa

- Methods returning an owned object
  - Name starts with alloc or new
  - Name contains copy
- All others imply not retuning an owned object

Core Foundation

- Functions whose names contain Create or Copy return an owned object
- Create is not a Cocoa convention!

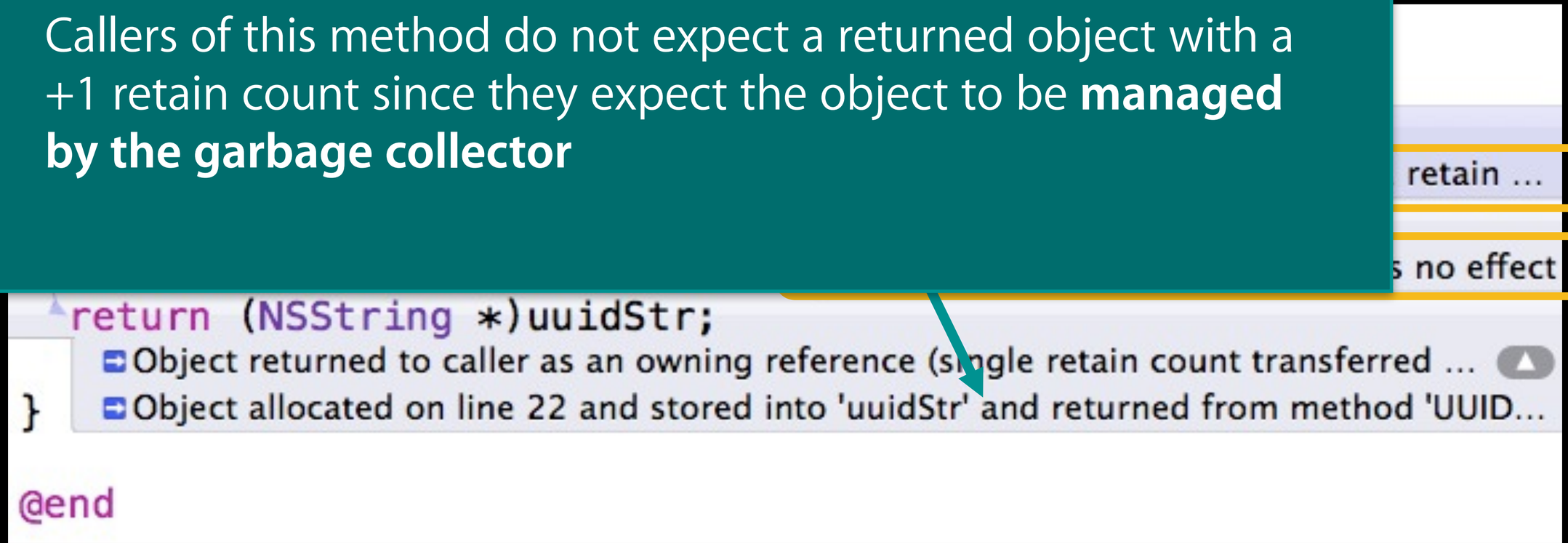
Diagnosing Leaks

```
void diagnosing_leaks() {

  NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithCapacity:4];
                              Method returns an Objective-C object with a +0 retain count (non-...
  [dict retain];              Reference count incremented. The object now has a +1 retain count

  [dict release];             Reference count decremented

  [dict retain];              Reference count incremented. The object now has a +1 retain count
}
  Object allocated on line 627 and stored into 'dict' is no longer referenced after this point and has a retain count o...
```

Leak When Using Garbage Collection

Object allocated on line 22 and stored into ‘uuidStr’ and returned from method ‘UUIDString:’ is potentially **leaked when using garbage collection**.

Callers of this method do not expect a returned object with a +1 retain count since they expect the object to be **managed by the garbage collector**



Overriding Cocoa Conventions

Only on Mac OS 10.6+

- Analyzer strictly enforces Cocoa conventions
  - Can cause false positives due to naming issues
- Rename your methods
  - Use Xcode refactoring feature
- Use ownership annotations

Annotation (Macro)	Header (10.6+)	Meaning
NS_RETURNS_RETAINED	Foundation.h (Cocoa)	Method returns an owned Objective-C object
CF_RETURNS_RETAINED	CoreFoundation.h	Method returns an owned Core Foundation object

```
@interface SQLDatabase : NSObject
...
-(SQLResult*)performQuery:(NSString*)inQuery;NS_RETURNS_RETAINED

-(SQLResult*)performQueryWithFormat:(NSString*)inFormat, ...;NS_RETURNS_RETAINED
...
@end
```

Overriding Cocoa Conventions

Only on Mac OS 10.6+

Annotation (Macro)	Header (10.6+)	Meaning
NS_RETURNS_RETAINED	Foundation.h (Cocoa)	Method returns an owned Objective-C object
CF_RETURNS_RETAINED	CoreFoundation.h	Method returns an owned Core Foundation object

Defining NS_RETURNS_RETAINED yourself...

```
#ifndef NS_RETURNS_RETAINED
#if defined(__clang__)
#define NS_RETURNS_RETAINED __attribute__((ns_returns_retained))
#else
#define NS_RETURNS_RETAINED
#endif
#endif
```

http://clang.llvm.org

Generic Checks

- Null dereferences
- Uses of uninitialized values
- Returning the address of a stack variable
- Sending a message to nil when the return value will be garbage
- Declaring a variable length array with zero element
- Undefined operations
- Divide-by-zero
- …

Null Dereference in ‘Wolfenstein3D for iPhone’

```
void vorbis_dsp_clear(vorbis_dsp_state *v){
  int i;
  if(v)
  {
    vorbis_info *vi = v->vi;

    codec_setup_info *ci = (codec_setup_info *) (vi ? vi->codec_setup : 0);

    private_state *b=(private_state *)v->backend_state;

    if(v->pcm)
    {
      for( i=0; i < vi->channels; i++)
        if(v->pcm[i])_ogg_free(v->pcm[i]);
      _ogg_free(v->pcm);
      if(v->pcmret)_ogg_free(v->pcmret);
    }
```

Assuming pointer value is null

Dereference of null pointer

Dead Stores

- What are they?
  - A category of dead code bugs
  - A value stored to a variable is never later read

```
x = foo();
... // no use of ‘x’
x = bar();
```

- Often indicate broken logic or optimization opportunities

Dead Store in 'WordPress for iPhone'

Example: Dead Store

```
BOOL everythingIsFine = YES;

if (![o doWork])
  everythingIsFine = NO;        Value stored to 'everythingIsFine' is never read

if (![o doMoreWork])
  everythingIsFine = NO;        Value stored to 'everythingIsFine' is never read

everythingIsFine = [o doLastWork];
```

Suppressing Dead Stores

- #pragma unused

```
int x = foo(); // no later use of ‘x’
...
#pragma unused(x)
```

- __attribute__((unused))

```
int y __attribute__((unused));
...
y = foo(); // no later use of ‘y’
```

- Both of these are standard annotations provided by GCC

Suppressing False Paths with Assertions

- Assertions tell the analyzer about your assumptions
  - Suppress false positives
  - Adds runtime checks for your assumptions

```
void loop_always_executes_at_least_once() {

  char *p = NULL;                          Variable 'p' initialized to a null pointer value

  for (unsigned i = 0, n = num_iterations(); i < n; ++i)
    p = getBuffer(i);


  *p = 1;                                  Dereference of null pointer
}
```

Suppressing False Paths with Assertions

- Assertions tell the analyzer about your assumptions
  - Suppress false positives
  - Adds runtime checks for your assumptions

```
void loop_always_executes_at_least_once() {

  char *p = NULL;

  for (unsigned i = 0, n = num_iterations(); i < n; ++i)
    p = getBuffer(i);

  assert(p != NULL);
  *p = 1;
}
```

Assertions + Data-Dependent Control-Flow

```
int y;                                  Variable 'y' declared without an initial value

if ([x tag] == 0) { y = 1; }
else if ([x tag] == 1) { y = 2; }
else if ([x tag] == 2) { y = 3; }

f(y);                                   Pass-by-value argument in function call is undefined
```

Assertions + Data-Dependent Control-Flow

```
int y;

if (x[tag] == 0) { y = 1; }
else if (x[tag] == 1) { y = 2; }
else { y = 3; }
  assert(x[tag] == 2);
f(y);
```

```
int y;

switch (x[tag]) {
  default: assert(0);
  case 0: y = 1; break;
  case 1: y = 2; break;
  case 2: y = 3; break;
}

f(y);
```

Always Analyze with Assertions Enabled!

- Assertions disabled in Release configuration
- Assertions enabled in Debug configuration
- Always analyze Debug configuration!
- Custom assertion handlers
  - Use noreturn and analyzer_noreturn attributes

http://clang.llvm.org

Analyzer Steadily Improving!

- 100% Open Source
- Please tell us about false positives!
- Please tell us about what bugs are important to you!

http://bugreporter.apple.com

http://clang.llvm.org

Summary

- Find many bugs faster and easier with static analysis
- Analyze both iPhone and Mac applications
- Tight integrated workflow
- Analyzer is 100% Open Source
- Join us today!

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Labs

Compiler Lab	OS X Lab D Wednesday 9:00AM
Improving Your Application with the Xcode Static Analyzer Lab	OS X Lab D Thursday 3:30PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09